§4 Amortized Analysis

- Motivation: Bit-Cost of Repeated Increment/Binary Add.
  - Amortized vs. Average vs. Worst-Case Analysis
- Fibonacci Heaps:
  - Relaxed Binomial Trees
  - *ExtractMin* and *DecreaseKey*
- Minimum Spanning Tree
  - Prim's Algorithm with Binomial vs. Fibonacci Heap
  - Kruskal's Algorithm: *Union-Find* data type
- Disjoint Set Data Structures
  - Fast and Slowly Growing Functions
  - Analysis of *Union-by-Weight*
  - Lazy Union-by-Rank with Path Compression

Bit-Cost of *Repeated* Increment

Cost := #bit flips when counting to N

```
000…000
000…001
000…010
000…011
000…100
000…101
……
111…111
```

One increment of value $<N$: $O(\log N)$

M inc.s cost $\leq \sum_{N<M} \log N = \Theta(M \cdot \log M)$

*over*estimate

Bit #0 flips M times,

bit #1 incurs $M/2$ flips, bit #2: $M/4$ flips

$\sum_M$ #flips $\leq 2M = \Theta(M)$:

amortized 2 flips "per" **`inc`**

"savings account"

Potential method of amortized analysis:

Let c_m := cost of m-th operation, $m=1..M$; devise Φ_m

Φ_m = #1s in counter after m-th op. =before $(m+1)$-st op.

$$\sum_{1\leq m\leq M} c_m/M \leq \max_m (c_m+\Phi_m-\Phi_{m-1}) + (\Phi_0-\Phi_M)/M \leq 2$$

Amortization: *Flat-Rate* Analysis

IT SERVICE FLATRATE

"savings account"

Potential method of amortized analysis:

Let c_m := cost of m-th operation, $m=1..M$; devise Φ_m

:= bank balance before m-th operation

$$\sum_{1\le m\le M} c_m/M \;\le\; \max_m \left(c_m+\Phi_m-\Phi_{m-1}\right) + (\Phi_0-\Phi_M)/M$$

Bit-Cost of Repeated INC & DEC

Cost := #bit flips

000…000
000…001
000…010
000…011
000…100
000…101
……
111…111

Examples: a) INC, INC, INC, …, INC

b) INC, INC, …, INC, DEC, DEC, …, DEC

c) INC, DEC, INC, DEC, …, INC, DEC

Answer: $\mathrm{O}(\log N)$ in the worst case

Amortized cost(N) := $1/N \cdot$ total (worst-case) cost of any N-element sequence of INCs and DECs

Repeated *Addition* of Binary Powers

Cost := #bit flips

Lemma: When adding a binary power 2^j to a natural number in binary, #bit flips + *difference* in #**1**s $\equiv$ **2**.

```
 01001111101011
+00000001000000
```

Problem: Add $2^{j_1}+2^{j_2}+\ldots+2^{j_n}$.

Theorem: Repeatedly (N times) adding powers of 2 incurs a total of $\leq 2N$ bit flips: has *amortized* cost 2.

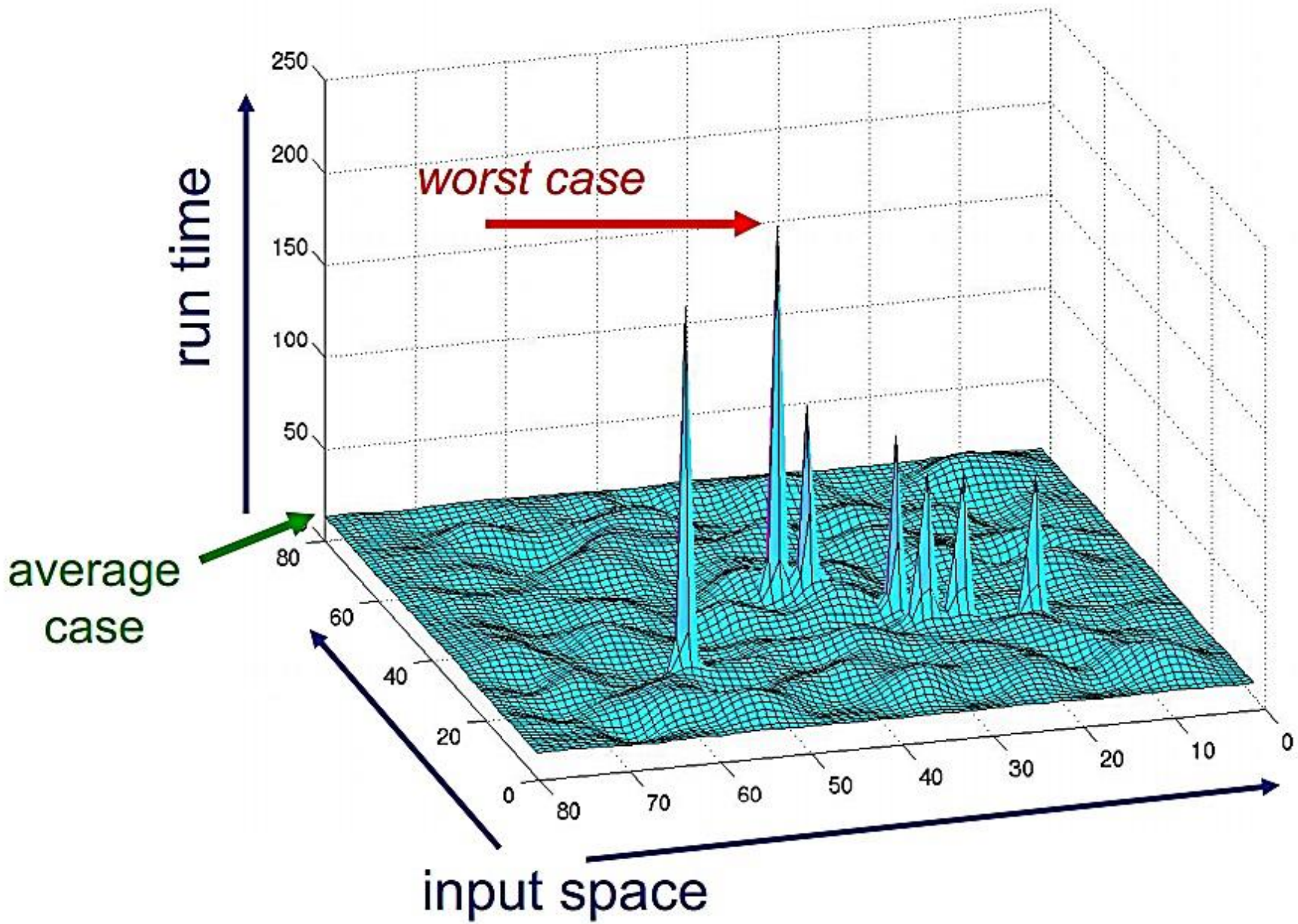
Potential method of amortized analysis:

Let c_m = cost of m-th operation, $m=1..M$; devise Φ_m

Φ_m = #1s in counter after m-th op. =before $(m+1)$-st op

$$\sum_{1\leq j\leq n} c_j/n \;\leq\; \max_j \left(c_j+\Phi_j-\Phi_{j-1}\right) + (\Phi_0-\Phi_n)/n \;\leq 2$$

Worst-Case vs. Average-Case Analysis



Amortized/Average/Worst Case

Example: Amortized #bit flips, counting to n in binary

Def: Fix data structure&algorithms for abstract data type: new initialized, then any N calls to method(s)/operations;
$\Rightarrow$ *total* worst-case cost $C(N)$ $\Rightarrow$ **amortized cost** $=C(N)/N$

inc **& dec** ?

- (Ordinary) worst-case: $\mathbf{max}_{|\underline{x}|\le n}\, c(\underline{x})$
- Average case: $\mathbf{sum}_{|\underline{x}|\le n}\, c(\underline{x}) \,/\, \#\{\, \underline{x}\colon |\underline{x}|\le n \,\}$
- Expected case: randomized algorithms (later).

pessimist

$c(\underline{x})$:= cost of *one* call on arguments/data contents $\underline{x}$

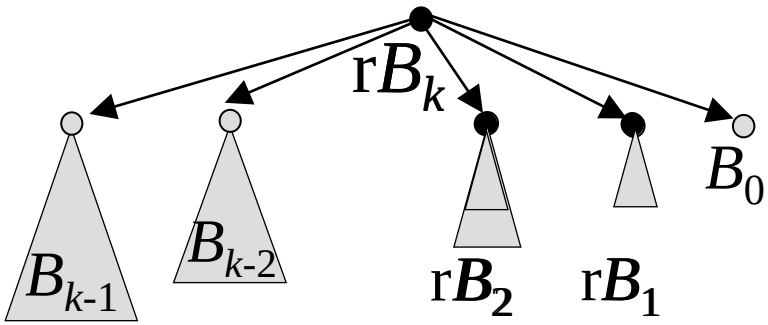
Potential method of amortized analysis:

Let c_m = cost of m-th operation, $m=1..M$; devise Φ_m

Conceive Φ_j such that right hand side is „small"

$$\sum_{1\le m\le M} c_m/M \;\le\; \mathbf{max}_m\,(c_m+\Phi_m-\Phi_{m-1}) + (\Phi_0-\Phi_M)/M$$

Relaxed Binomial Trees



A *relaxed* binomial tree of order $k \geq 1$ is a root with k children: relaxed binomial trees of *distinct* orders.

Merge

Prune

from *un*marked parent: add mark to parent

Prove by strong induction:

$$\#\text{nodes} \geq 1 + F_1 + F_2 + \ldots + F_{k-1} + F_k = F_{k+2} \geq \varphi^k$$

$\varphi = (1+\sqrt{5})/2 > 1.6$

$F_0=0,\ F_1=1,\ F_{k+1}= F_k + F_{k-1}$

Lemma: A *relaxed* binomial tree of order k has $\geq F_{k+2} \geq \Omega(1.6^k)$ nodes

Quiz

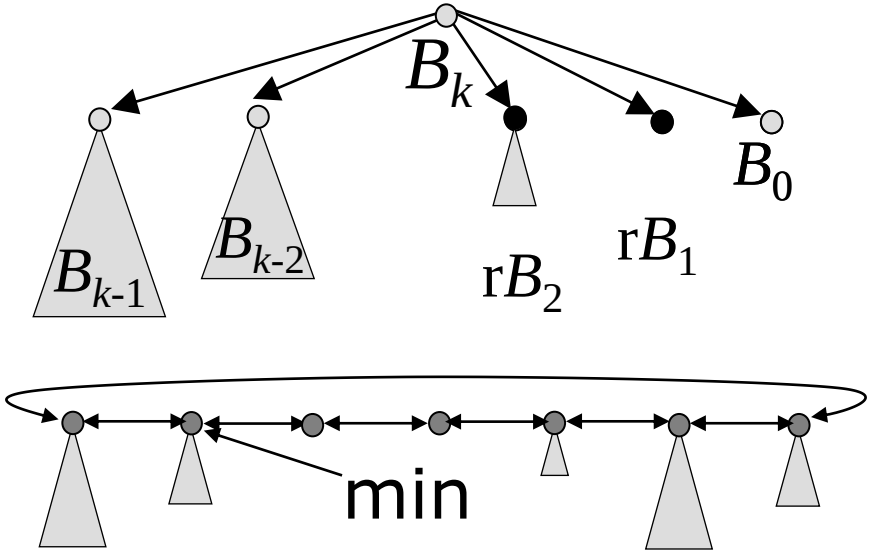
a)
Prove by strong induction:

$$1 + F_1 + F_2 + \ldots + F_{k-1} + F_k = F_{k+2}$$

b) Prove:

$$\sum_{1 \le m \le M} c_m / M \;\le\; \max_m \left(c_m + \Phi_m - \Phi_{m-1}\right) + \left(\Phi_0 - \Phi_M\right)/M$$

Fibonacci Heaps



A *relaxed* binomial tree of order $k \geq 1$ is a marked root with k children: relaxed binomial *distinct* orders $< k$.

Operations with amortized costs:

ExtractMin: $O(\log n)$

DecreaseKey: $O(1)$

Merge: $O(1)$

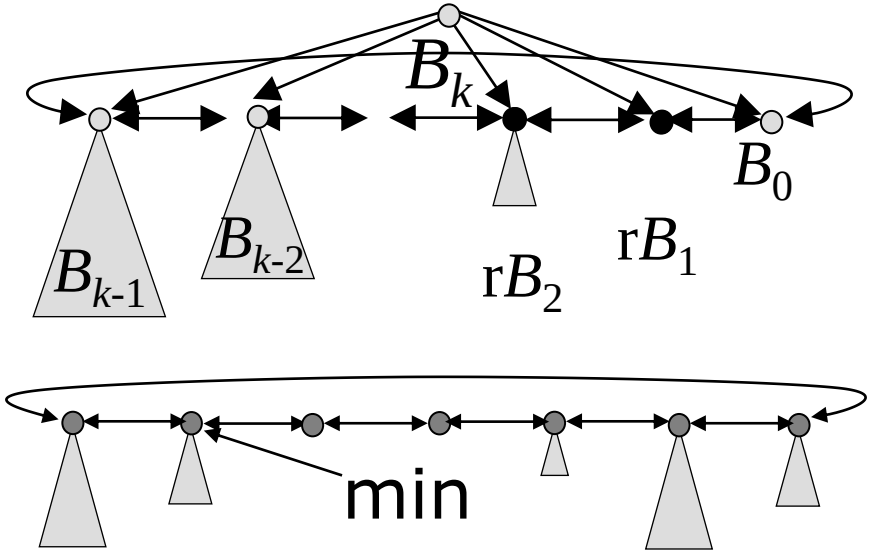
InsertKey: $O(1)$

Create: $O(1)$

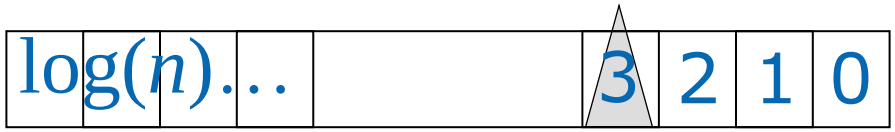
A Fibonacci Heap is a list of t heap-ordered relaxed binom. trees with pointer to the min.

Lemma: A *relaxed* binomial tree of n nodes has order $k \leq O(\log n)$

ExtractMin: O(log *n*) Amortized



A *relaxed* binomial tree of order $k \geq 1$ is a root with k children: relaxed binomial trees of *distinct* orders.



1. Delete target of min.pointer
2. Merge Fibonacci Heaps.
3. **Consolidate** list s.t. relaxed binom.trees have distinct orders k

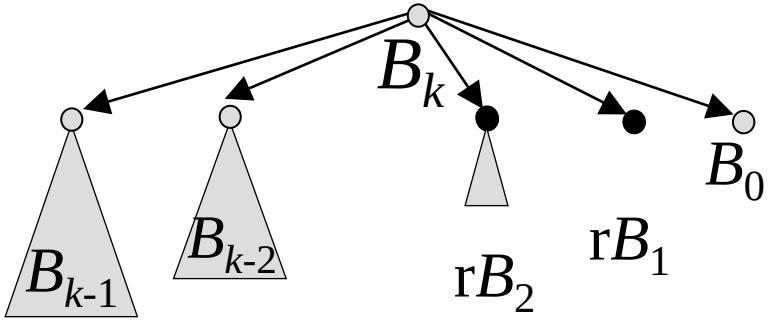
Cmp. adding t binary powers: tot.cost $O(t)$

A Fibonacci Heap is a list of t heap-ordered relaxed binom. trees with pointer to the min.

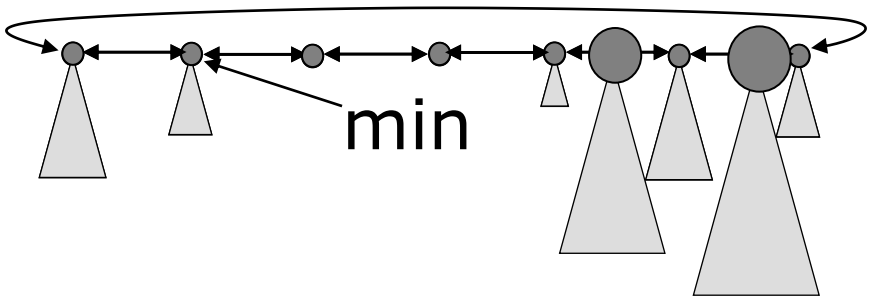
Conceive $\Phi := \Theta(t) \Rightarrow \Phi_0 = 0 \leq \Phi_n,\ c_j + \Phi_j - \Phi_{j-1} \leq O(\log n)$

$$\sum_{1 \leq j \leq n} c_j/n \leq \max_j (c_j + \Phi_j - \Phi_{j-1}) + (\Phi_0 - \Phi_n)/n$$

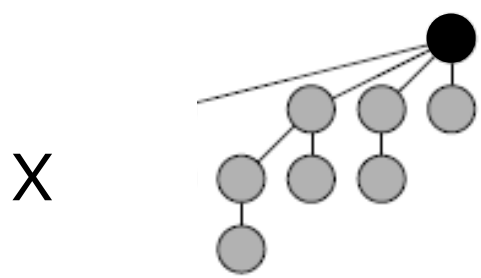
DecreaseKey: O(1) Amortized



A *relaxed* binomial tree of order $k \geq 1$ is a root with k children: relaxed binomial trees of *distinct* orders.



- cut subtree
- **mark** parent
- if already **marked**:
- reset, cut & cascade up



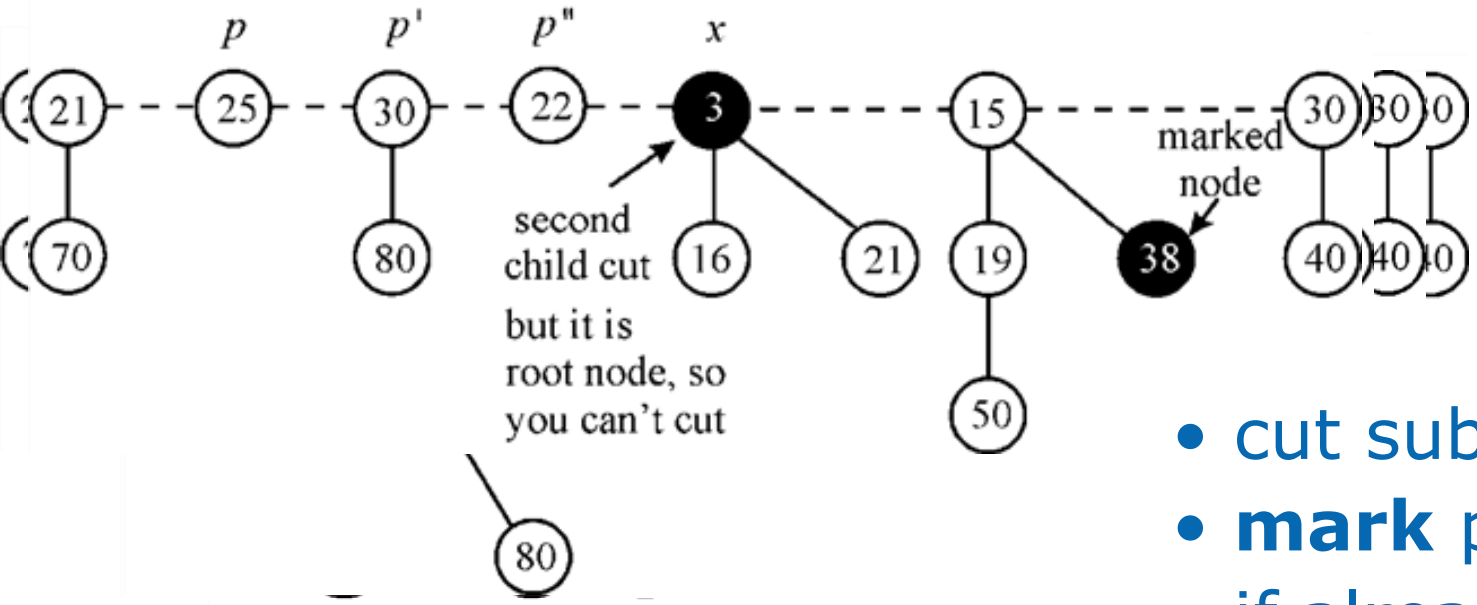
m := #marks

A Fibonacci Heap is a list of t heap-ordered relaxed binom. trees with pointer to the min.

Conceive $\Phi := \Theta(t+2m)$ $\Phi_0 = 0 \leq \Phi_n$, $c_j + \Phi_j - \Phi_{j-1} \leq O(1)$

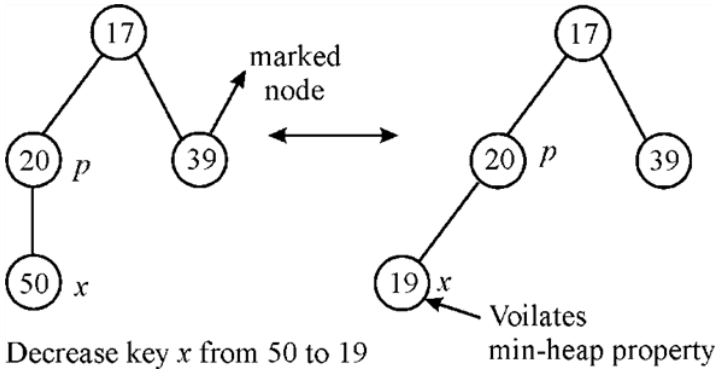
$$\sum_{1 \leq j \leq n} c_j/n \leq \max_j (c_j + \Phi_j - \Phi_{j-1}) + (\Phi_0 - \Phi_n)/n$$

Cuts, Marks, and Cascading

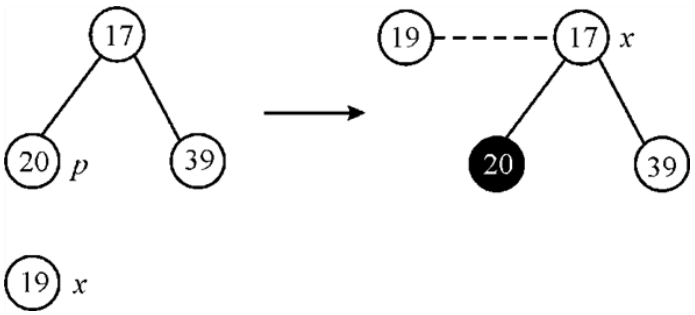


(decrease key of x from 42 to 21)

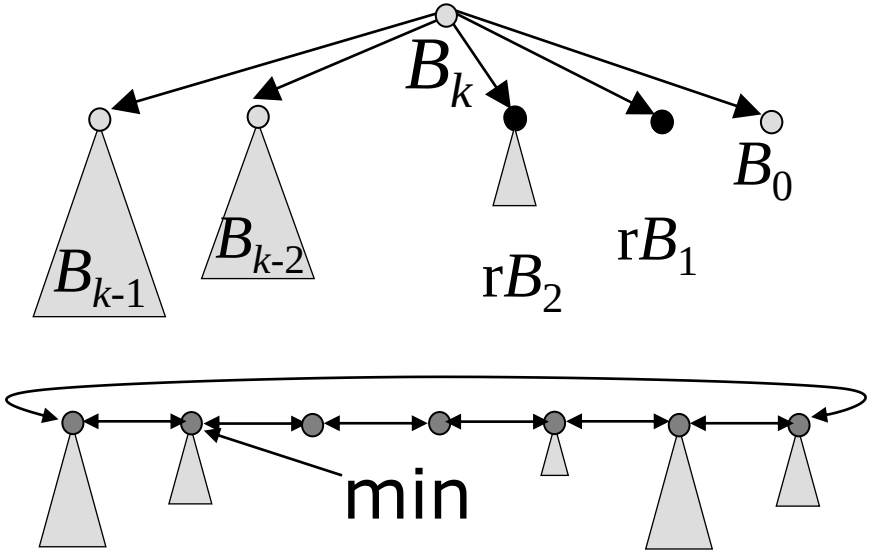
- cut subtree
- **mark** parent
- if already **marked**:
- reset, cut
  & cascade up



Decrease key x from 50 to 19



Fibonacci Heap: Summary



A *relaxed* binomial tree of order $k \geq 1$ is a marked root with k children: relaxed binomial *distinct* orders $<k$.

lazy consolidation

Amortized analysis using **two** "bank accounts":

Priority Queue datatype, **amortized** costs**:**

ExtractMin: $O(\log n)$

DecreaseKey: $O(1)$

Merge: $O(1)$

InsertKey: $O(1)$

Create: $O(1)$

A Fibonacci Heap is a list of t heap-ordered relaxed binom. trees with pointer to the min.

one with balance t and

one with balance $2m$ (#marks)

Minimum Spanning Tree

Tree: connected (undirect.) graph without cycles

weighted undir. graph (V,E,w) where $E \subseteq V \times V$ and $w: E \to \mathbb{R}_+$ s.t. $(u,v) \in E \Rightarrow (v,u) \in E \ \wedge \ w(u,v) = w(v,u) < \infty$

MST (Minimum Spanning Tree) of (V,E,w):

a (i) *connected* subset F of the edges E which

(ii) includes *all* vertices and

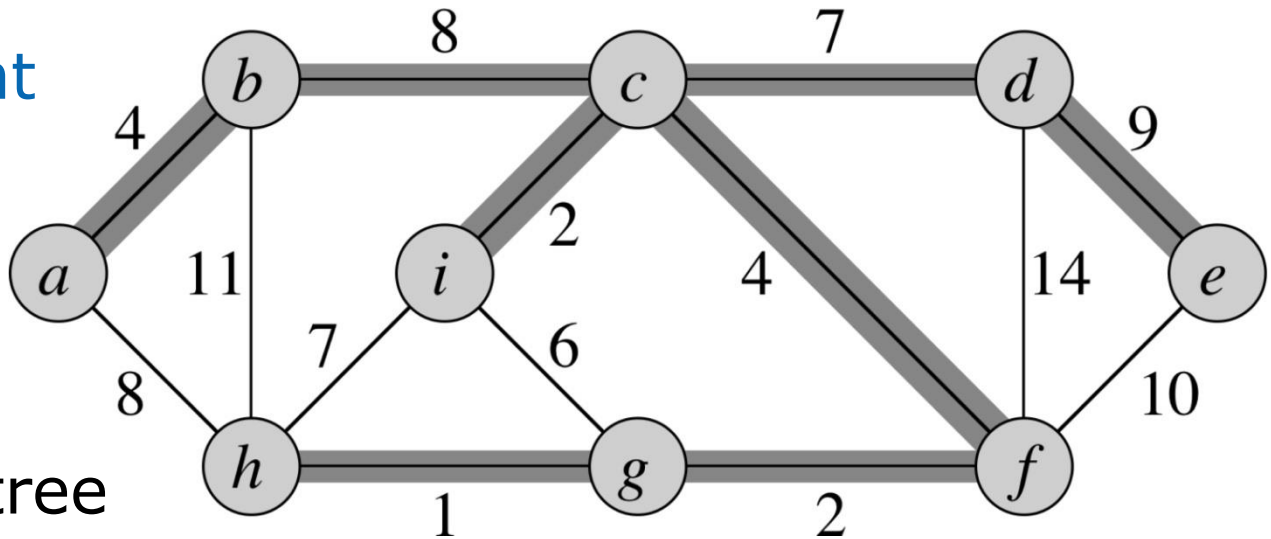
(iii) has *least* weight

$w(F) = \sum_{(u,v) \in F} w(u,v)$

among all $F' \subseteq E$

satisfying (i)+(ii)

$\Rightarrow$ automatically a tree



Prim's Algorithm for MST

// $w : E \subseteq V \times V \to \mathbb{R}_+$ edge weights

$F := \{\}$; FOREACH $v \in V$ DO

v.dist := ∞ ; v.neighbor := NIL; DONE

$Q := V$; WHILE not Q.**isEmpty** DO

u := Q.**ExtractMin**; u.dist := 0

IF u.neighbor $\neq$ NIL THEN

$F := F \cup \{ (u.\text{neighbor}, u) \}$;

FOREACH v adjacent to u DO

IF $w(u,v) < v$.dist THEN

v.neighbor := u;

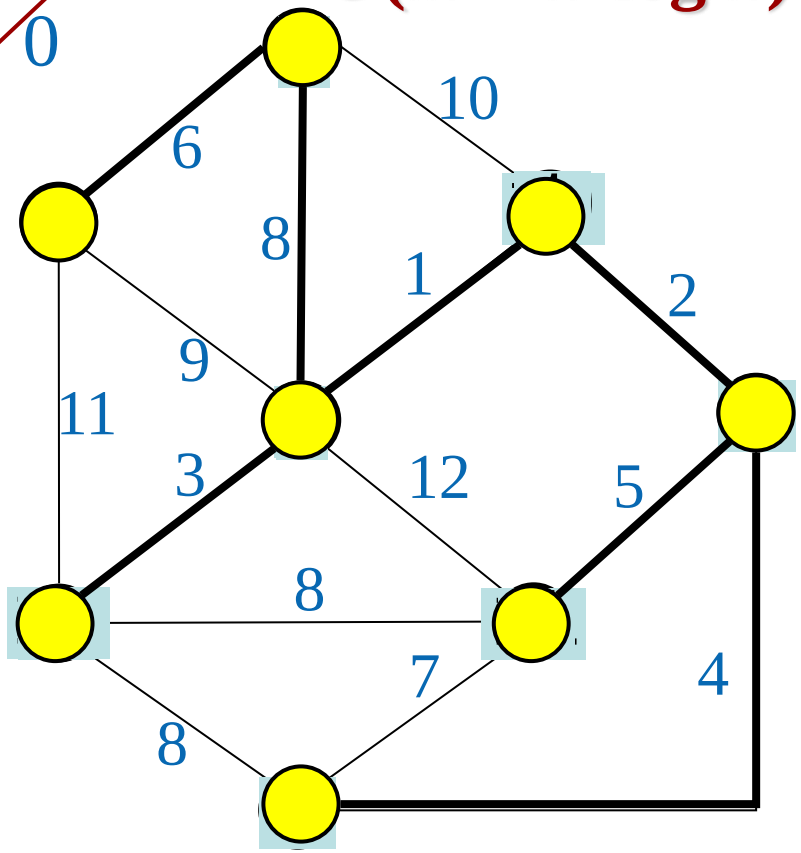
Q.**decreaseKey**$(v, w(u,v))$;

END;

$n = |V|$ times

$m = |E| \geq n$ times

Fibonacci Heap:
$O(m + n \cdot \log n)$



Kruskal's Algorithm for MST

// $w : E \subseteq V \times V \to \mathbb{R}_+$ edge weights

$n = |V|$ times

$F := \{\}$; FOREACH $v \in V$ DO **MakeSet**(v); // singletons

FOREACH edge $(u,v) \in E$ in order of increasing weight DO

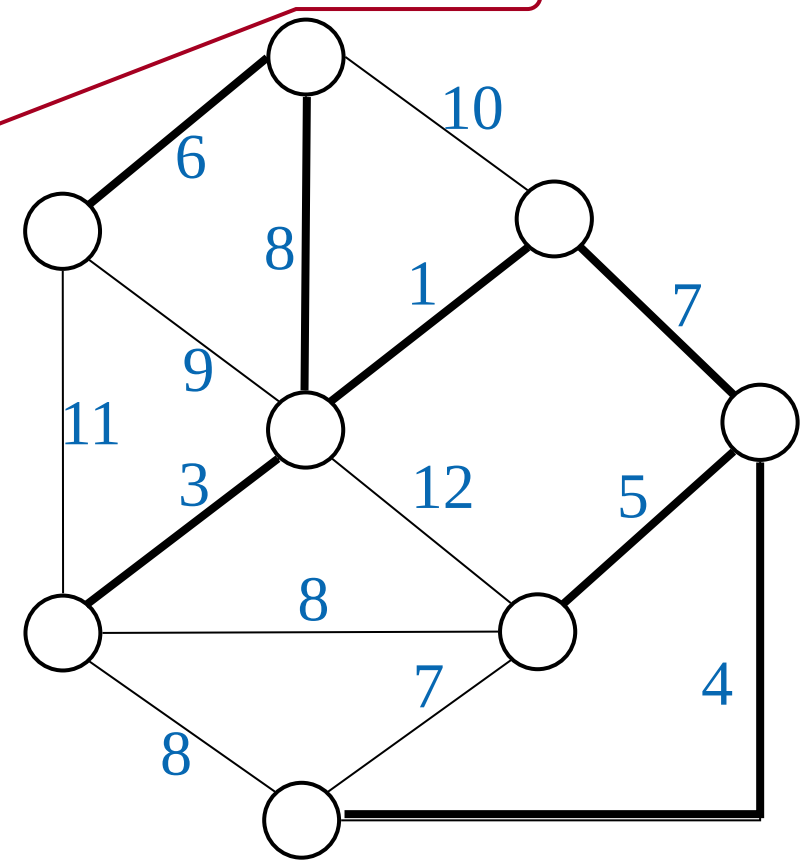
IF u and v do NOT belong to the same subset

THEN $F := F \cup \{ (u,v) \}$;

Union(u,v);

END;

FindSet$(u) \neq$ **FindSet**(v),
$m = |E|$ times



Disjoint-Set Data Structure:

$O(m \cdot \alpha(m)) < o(m \cdot \log^*(m))$

Prim's Algorithm with
Fibonacci Heap: $O(m + n \cdot \log n)$

Fast&Slowly Growing Functions

2^n exponential $\qquad \log N := \min\{\, n : 2^n \geq N \,\}$

2^{2^n} *doubly* exponential $\qquad \operatorname{loglog} N := \min\{\, n : 2^{2^n} \geq N \,\}$

…

$2^{2^{\cdot^{\cdot^{2}}}}$ *tower* of height n

aka *tetration* $2\uparrow\uparrow n = 2^{2\uparrow\uparrow(n-1)}$

$\log^*(N)$ = #iterations of log before argument ≤ 1
$= 1 + \log^*(\log N),\ N>1$

Quiz: $\log(2^{64})=?$ $\qquad \operatorname{loglog}(2^{128})=?$ $\qquad \log^*(2^{512})=?$

Wilhelm Ackermann (1896—1962) function

$A_0(n)=n+2,\quad A_{k+1}(0)=A_k(1),\quad A_{k+1}(n+1) = A_k(\,A_{k+1}(n))$

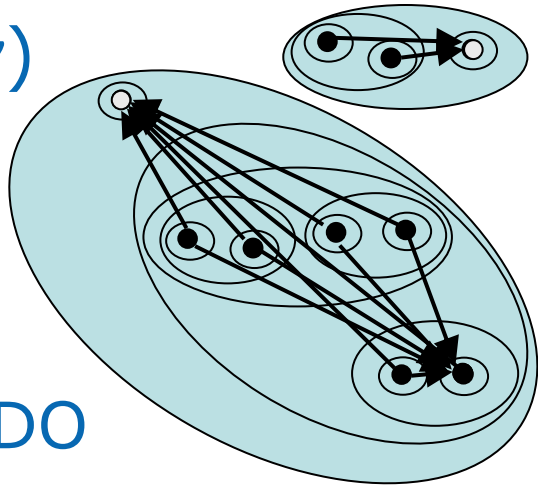
$A_1(n) = 2n+3,\quad A_2(n) = 2^{n+3}-3,\quad A_3(n) = 2\uparrow\uparrow(n+3)-3$

Inverse Ackermann $\alpha(N) = \min\{\, n : A_n(n) \geq N \,\}$

Disjoint-Set Data Structure

MakeSet(x) **FindSet**(x) **Union**(x,y)

return a unique "handle" to the set



Recall Kruskal MST of $G=(V,E,w)$

FOREACH $(u,v)\in E$ in increasing weight w DO

IF **FindSet**(u)≠**FindSet**(v)

THEN $T := T \cup \{ (u,v) \}$; **Union**(u,v) ;

amortized efficiency?

Naïve realization as forest of depth 1:

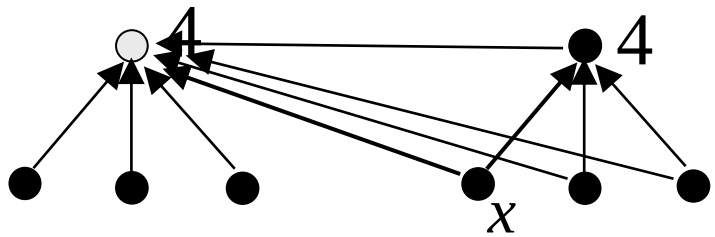
N times alternate **MakeSet+Union** → total time $\Omega(N^2)$

Union-by-weight heuristic: attach smaller to larger tree

Theorem: This yields <u>total</u> time $O(m+n\cdot\log n)$ for any sequence of n calls **MakeSet** & m calls **FindSet/Union**

Analysis of Union-by-Weight

MakeSet(x) **FindSet**(x) **Union**(x,y)



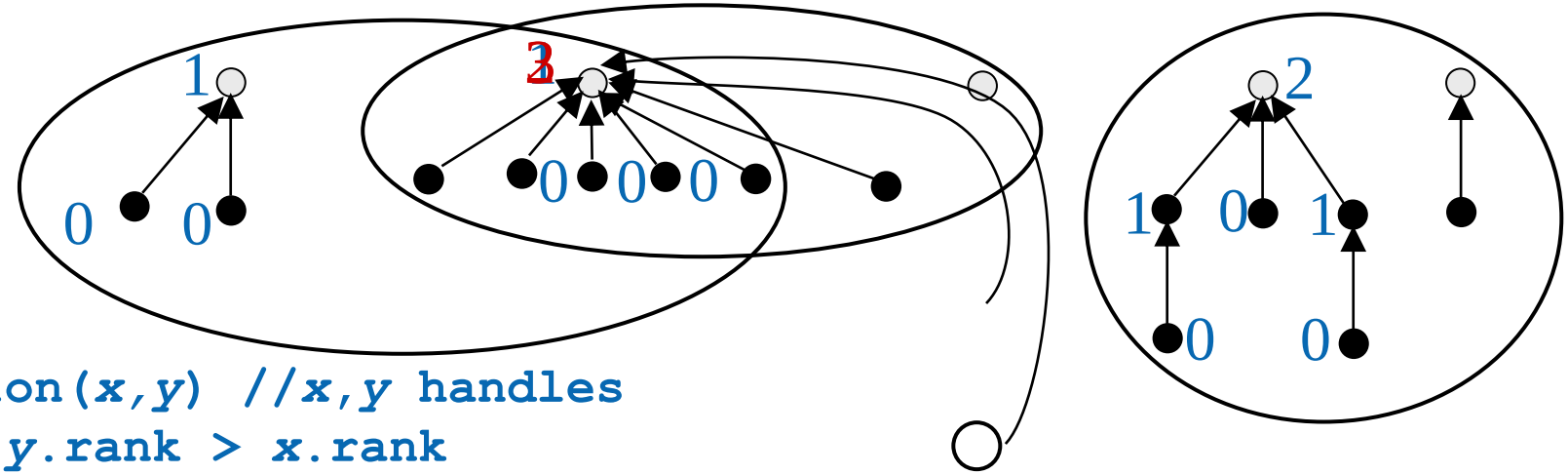
Observation: A node's link gets *updated* only when its set is combined with one of <u>larger or equal</u> size.

So *update* #k can occur only <u>after</u> $n \geq 2^k$ calls **MakeSet**.

MakeSet, **FindSet**: $O(1)$, **Union** ?

Theorem: This yields <u>total</u> time $O(m+n\cdot\log n)$ for any sequence of n calls **MakeSet** & m calls **FindSet/Union**

Lazy Union by Rank, Path Compression




```
Union(x,y) //x,y handles
if y.rank > x.rank
 then attach x to y
else attach y to x;
 if x.rank == y.rank
  then x.rank++; fi; fi;
```

```
function FindSet(x): if x.parent≠NIL
 then x.parent := FindSet(x.parent);
return(x.parent);
```

~~Naïve~~ realization as forest of <u>unbounded</u> depth:

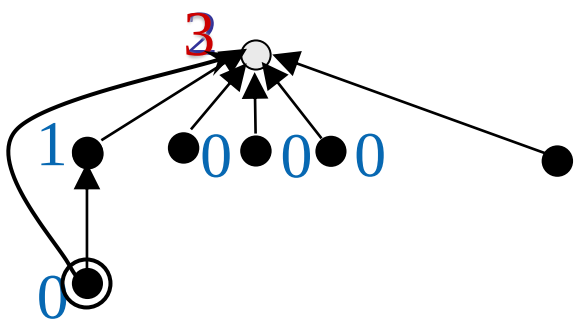
MakeSet, **Union**: $O(1)$, **FindSet** ? *Path compression*

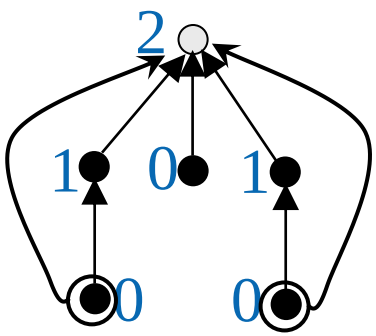
Lazy Union-by-<u>rank</u>: attach lower rank tree to higher

Theorem: This algorithm makes n **MakeSet** and m **FindSet** and **Union** calls run in <u>total</u> time $O(m \cdot \alpha(n))$.

Analysis of Path Compression I

semi- dynamic





Re-order: First **`MakeSets`**, then **`Unions`**, then **`Finds`**

a) Rank = height

b) Compress. *partial* path P

i) $\mathrm{top}(P)$ becomes nodes' parent

ii) some ancestor of $\mathrm{top}(P)$ becomes parent

$\mathrm{cost}(P)$:=
:= #nodes changing parents
= length -1 for type **(i)**
:= 0 for type (ii)

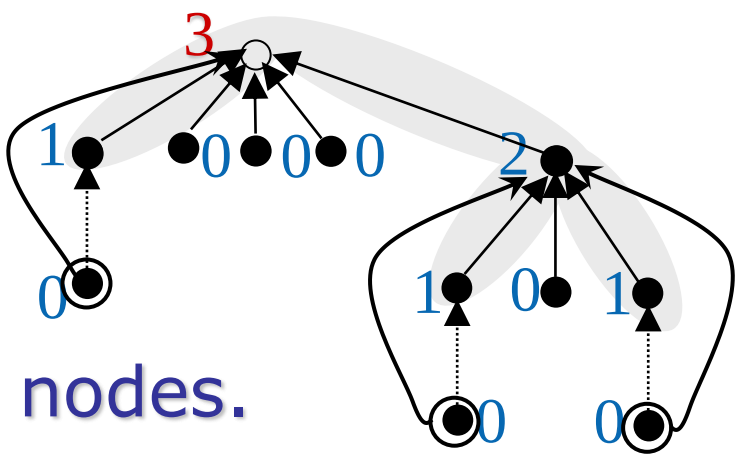
Sequence $\mathcal{P}=(P_1,..,P_m)$ of partial compress. paths

$|\mathcal{P}|$:= $\#P\in\mathcal{P}$ of type **(i)**

$\mathrm{cost}(\mathcal{P})$:= add costs of all $P\in\mathcal{P}$

Cost of m Partial Path Compressions

semi- dynamic



Fix set $T \subseteq V$ containing all its ancestors.

Split every path P into:

- part $P_{T'}$ *out*side of T
- part P_T *in*side of T.

(Each part possibly $\varnothing$)

Fix a forest with $n = |V|$ nodes.

Lemma:

a) $\mathrm{cost}(\mathcal{P}) \leq \mathrm{cost}(\mathcal{P}_T) + \mathrm{cost}(\mathcal{P}_{T'}) + |\mathcal{P}_T| + |T'|$

b) $|\mathcal{P}_T| + |\mathcal{P}_{T'}| \leq |\mathcal{P}|$

$\mathrm{cost}(P)$:=
:= #nodes changing parents
= length -1 for type **(i)**
:= 0 for type (ii)

Compress. p*artial* path P

i) $\mathrm{top}(P)$ becomes nodes' parent

ii) some <u>ancestor</u> of $\mathrm{top}(P)$ becomes parent

$|\mathcal{P}|$:= $\#P \in \mathcal{P}$ of type **(i)**

Sequence $\mathcal{P}=(P_1,..,P_m)$ of partial compress. paths

$\mathrm{cost}(\mathcal{P})$:= add costs of all $P \in \mathcal{P}$

First Cost Estimate

$|T|:=n/2 \;\Rightarrow\; \mathrm{cost}(\mathcal{P}) \le (|\mathcal{P}|+n/2)\cdot\log_2 n$

Proof by induction + lemma: $\mathrm{cost}(\mathcal{P})$
$\le \mathrm{cost}(\mathcal{P}_T) + \mathrm{cost}(\mathcal{P}_{T'}) + |\mathcal{P}_T| + |T'|$
$\le (|\mathcal{P}_T|+n/4)\cdot\log_2 n/2$
$+ (|\mathcal{P}_{T'}|+n/4)\cdot\log_2 n/2$
$+ |\mathcal{P}| + n/2$
$\le (|\mathcal{P}|+n/2)\cdot\log_2 n$ ■

Fix set $T\subseteq V$ containing all its ancestors.

Split every path P into:

- part $P_{T'}$ *out*side of T
- part P_T *in*side of T.

(Each part possibly $\varnothing$)

Did not take into account *union-by-rank*!

Lemma:

a) $\mathrm{cost}(\mathcal{P}) \le \mathrm{cost}(\mathcal{P}_T) + \mathrm{cost}(\mathcal{P}_{T'}) + |\mathcal{P}_T| + |T'|$

b) $|\mathcal{P}_T| + |\mathcal{P}_{T'}| \le |\mathcal{P}|$

Forest $\mathcal{F}$ with $n = |V|$ nodes.

Sequence $\mathcal{P}=(P_1,..,P_m)$ of partial compress. paths

Rank Forests Divide/Conquer

Def: In a *rank forest* of rank $r\in\mathbb{N}$, each node x has $\mathrm{rank}(x)\le r$ and children of <u>all&only</u> ranks $<\mathrm{rank}(x)$.

Lemma: Let $\mathcal{F}$ a rank forest of rank r.

For $s<r$ consider $T := \{\text{nodes of rank}>s\}$

a) $\mathcal{F}\cap T$ is a rank forest of rank $\le r-s-1$.

b) $\mathcal{F}\cap T'$ is a rank forest of rank $\le s$.

c) $|T|\le |V| / 2^{s+1}$;

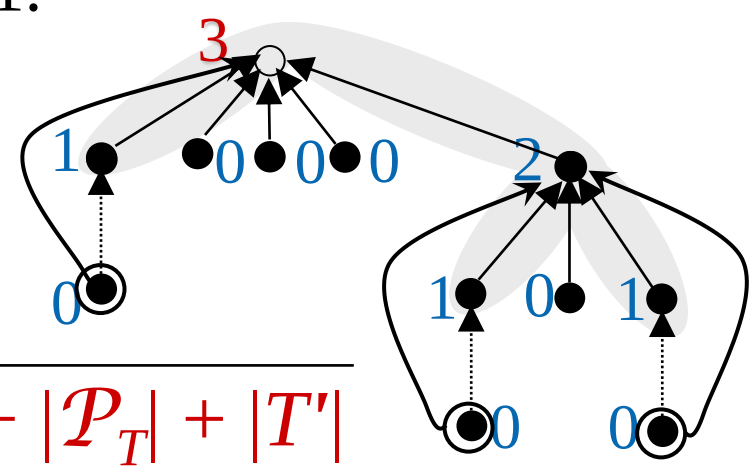
$\mathrm{rank}(x)=s \;\Rightarrow\;$ subtree of x has size $\ge 2^s$.

Fix set $T\subseteq V$ containing all its ancestors.

Split every path P into:

- part $P_{T'}$ *out*side of T
- part P_T *in*side of T.

(Each part possibly $\varnothing$)



a) $\mathrm{cost}(\mathcal{P}) \le \mathrm{cost}(\mathcal{P}_T) + \mathrm{cost}(\mathcal{P}_{T'}) + |\mathcal{P}_T| + |T'|$

b) $|\mathcal{P}_T| + |\mathcal{P}_{T'}| \le |\mathcal{P}|$

Forest $\mathcal{F}$ with $n = |V|$ nodes.

Sequence $\mathcal{P}=(P_1,..,P_m)$ of partial compress. paths

Union-by-Rank Analyses

Let $s := \log r$, $t := \log s$ $\Rightarrow$

$\varphi(n,m,r)$:= worst-case cost of m partial compression paths on any rank forest of rank r with n nodes.

$$\mathrm{cost}(\mathcal{P}_{V'_s}) \le n + \mathrm{cost}(\mathcal{P}_{V'_t}) + (|\mathcal{P}_{V'_s}| - |\mathcal{P}_{V'_t}|) + n$$

Previous: $\boxed{\varphi(n,m,r) \le (m+n/2)\cdot\log_2 n}$

$$\varphi(n,m,r) = \mathrm{cost}(\mathcal{P})$$

$$\le n + \mathrm{cost}(\mathcal{P}_{V'_s}) + (|\mathcal{P}_V| - |\mathcal{P}_{V'_s}|) + n \quad \le O(m+n\cdot\log\log n)$$

$$\le 2n + \mathrm{cost}(\mathcal{P}_{V'_t}) + (|\mathcal{P}_V| - |\mathcal{P}_{V'_t}|) + 2n \quad \le O(m+n\cdot\log\log\log n)$$

$$\le k\cdot n + \mathrm{cost}(\mathcal{P}_{V'_k}) + (|\mathcal{P}_V| - |\mathcal{P}_{V'_k}|) + k\cdot n \quad \le O(m+k\cdot n\cdot\log^{(k)} n)$$

$k := \log^* n$

$$\varphi(n,m,r) \le O(m+n\cdot\alpha(n))$$

a) $\mathrm{cost}(\mathcal{P}) \le \mathrm{cost}(\mathcal{P}_T) + \mathrm{cost}(\mathcal{P}_{T'}) + |\mathcal{P}_T| + |T'|$

b) $|\mathcal{P}_T| + |\mathcal{P}_{T'}| \le |\mathcal{P}|$

Forest $\mathcal{F}$ with $n = |V|$ nodes.

Sequence $\mathcal{P} = (P_1,..,P_m)$ of partial compress. paths

§4 Summary

- Motivation Bit-Cost of Repeated Increment/Binary Add.
  - Amortized vs. Average vs. Worst-Case Analysis
- Fibonacci Heaps:
  - Relaxed Binomial Trees
  - *ExtractMin* and *DecreaseKey*
- Minimum Spanning Tree
  - Prim's Algorithm with Binomial vs. Fibonacci Heap
  - Kruskal's Algorithm: *Union-Find* data type
- Disjoint Set Data Structures
  - Fast and Slowly Growing Functions
  - Analysis of *Union-by-Weight*
  - Lazy Union-by-Rank with Path Compression